COMMON LISP: *An Interactive Approach*
by Stuart C. Shapiro
Published by Computer Science Press, An Imprint of W. H. Freeman and Company • New York, 1992.

ERRATA[1]

page xviii, line 3: Change “This” to “It”.

page 21, line -6: Change

```
> (+ 12 4) 16
```

to

```
> (+ 12 4)
16
```

page 21, line -1: Move line to top of page 22.

page 26, line -3: Change

$$(25 + 30) \times 15/2$$

to

$$(25 + 30) \times \frac{15}{2}$$

page 85, line 16: Change “function” to “number”.

page 86, line 18: Change “return” to “be”.

page 102, line -3: Change “`'()`” to “`nil`”.

page 121, line -13: Change “`((+ 3 5) - 6 * 8))`” to “`((+ 3 5) - 6 * 8)`”.

page 121, line -7: Change “`combine-expression`” to “`enclose-expression`”.

page 130, lines 1–7: Change indentation from

```
(defun bstreep (tree)
  "A bstree is either an element, or a three-member list,
   the first of which is an element."
  (or (typep tree 'util:element)
      (and (listp tree)
  (= (length tree) 3)
  (typep (first tree) 'util:element))))
```

to

```
(defun bstreep (tree)
  "A bstree is either an element, or a three-member list,
   the first of which is an element."
  (or (typep tree 'util:element)
      (and (listp tree)
           (= (length tree) 3)
           (typep (first tree) 'util:element))))
```

page 143, line 17: Change “using backquote” to “using the backquote”.

[1] Thanks to J. Terry Nutter for significant contributions to this list.

page 145, Exercise 19.2: Change the numbering from

```
a. (eval 5)
b. (eval a)
c. (eval 'a)
d. (eval ''a)
e. (eval (first (a))) (eval (first '(a)))
f. (eval '(first (a)))
g. (eval '(first '(a)))
h. (eval (cons 'first '((a))))
i. (eval (cons 'first '('(a))))
```

to

```
a. (eval 5)
b. (eval a)
c. (eval 'a)
d. (eval ''a)
e. (eval (first (a)))
f. (eval (first '(a)))
g. (eval '(first (a)))
h. (eval '(first '(a)))
i. (eval (cons 'first '((a))))
j. (eval (cons 'first '('(a))))
```

page 147, line -1: Change “on” to “in”.

page 153, lines 8–16: Change the indentation from

```
(defun scalar-add1 (vector)
  "VECTOR is a list of numbers.
   Returns a list just like it,
   except that each number in it is incremented by 1."
  (check-type vector list)
    (typecase vector
    (null '())
    (cons (cons (1+ (first vector))
                (scalar-add1 (rest vector))))))
```

to

```
(defun scalar-add1 (vector)
  "VECTOR is a list of numbers.
   Returns a list just like it,
   except that each number in it is incremented by 1."
  (check-type vector list)
  (typecase vector
    (null '())
    (cons (cons (1+ (first vector))
                (scalar-add1 (rest vector))))))
```

page 154, line -8: Change “`scalar-add1`, then, is” to “`scalar-add1` is”.

page 160, line 9: Change “`\&rest`” to “`&rest`”.

page 161, line -13: Change “Lisp’s” to “Lisp”.

page 171, line -9: Change “`(macroexpand '(pluslist-f (+ 3 4) 5))`” to “`(macroexpand '(pluslist-m (+ 3 4) 5))`”.

page 176, line -8: Insert a little space to change “*symbol*’s” to “*symbol* ’s”.

page 184, line -1: Change “`(setq x 3)`” to “`(setf x 3)`”.

page 207, line 10: Change “`12)))`” to “`12))`”.

page 207, line 17: Change “*forms*” to “*statements*”.

page 207, line 18: Change “*form*” to “*statement*”.

page 210, line -1: Change “`list:dolist`” to “`lisp:dolist`”.

page 217, line 13: Change “evaluate form” to “evaluate a form”.

page 225, line -10: Change

```
(format t " &I heard you say    A   " sentence)
```

to

```
(format t "~&I heard you say ~{~A~}" sentence)
```

page 238, line 15: Change “30.18 Define” to “30.18 *(i)* Define”.

page 238, line -4: Change “`'numberp`” to “`#'numberp`”.

page 248, line 1: Change “can be any list” to “can be any Lisp”.

page 260, line -6: Change “the generic method” to “the generic function”.

page 262, lines 1–7 Move line 7 up, changing

```
> (defmethod < ((x number) (y number))
    (lisp:< x y))
> (< 3 5)
T
> (< 5 3)
NIL
#<method (TICLOS:METHOD < (NUMBER NUMBER))>
```

to

```
> (defmethod < ((x number) (y number))
    (lisp:< x y))
#<method (TICLOS:METHOD < (NUMBER NUMBER))>
> (< 3 5)
T
> (< 5 3)
NIL
```

page 262, line 8: Change “generic method” to “generic function”.

page 280, line 20: Change “`(princ '_)`” to “`(princ '_ stream)`”.

page 297, lines 1 – 11: Change

```
17.34 (defun enclose-expression (expr)
      "EXPR is a list representing an arithmetic
       expression (using only the operators + and -)
       in normal infix notation.
       Returns a list whose one member is EXPR
       transformed into Cambridge Prefix Notation."
      (check-type expr list)
      (cond ((< (length expr) 3) expr)
            (t (combine-expr
                 (second expr) (first expr)
                 (enclose-expression (nthcdr 2 expr))))))
```

to

```
17.34 (defun enclose-expression (expr)
        "EXPR is a list representing an arithmetic
         expression (using only the operators + and -)
         in normal infix notation.
         Returns a list whose one member is EXPR transformed
         into Cambridge Prefix Notation."
        (check-type expr list)
        (cond ((< (length expr) 3) expr)
              (t (enclose-expression
                  (combine-expr
                   (second expr) (first expr) (nthcdr 2 expr))))))
```

page 304, lines 13 – 28: Change

```
(defun substitute (pat subs)
  "Returns a tree like PAT,
   but with every variable that is bound in
   the substitution SUBS
   replaced by the term it is bound to."
  (check-type subs list)
  (cond ((atom pat)
         (if (and (variablep pat) (boundp pat subs))
             (bound-to pat subs)
             pat))
        ((and (svariablep (first pat))
              (boundp (first pat) subs))
         (append (bound-to (first pat) subs)
                 (substitute (rest pat) subs)))
        (t (cons (substitute (first pat) subs)
                 (substitute (rest pat) subs)))))
```

to

```
(defun substitute (pat subs)
  "Returns a tree like PAT,
   but with every variable that is bound in
   the substitution SUBS
   replaced by the term it is bound to."
  ;; pat cannot be a sequence variable because there might be no
  ;; environment to splice its value into.
  (check-type pat (not (satisfies svariablep)))
  (check-type subs list)
  (cond ((atom pat)
         (if (and (variablep pat) (boundp pat subs))
             (bound-to pat subs)
           pat))
        ((and (svariablep (first pat))
              (boundp (first pat) subs))
         (append (bound-to (first pat) subs)
                 (substitute (rest pat) subs)))
        (t (cons (substitute (first pat) subs)
                 (substitute (rest
                             pat) subs)))))
```

page 310, lines 7 – 13: Change

```
33.5 (defmethod < ((x t) (y t))
       (check-type x
           (or character number symbol string))
       (check-type x
           (or character number symbol string))
       (member (type-of y)
               (member
                (type-of x)
                '(character number symbol string))))
```

to

```
33.5 (defmethod < ((x t) (y t))
       (check-type x
           (or character number symbol string))
       (check-type x
           (or character number symbol string))
       (member (gentype-of y)
               (member
                (gentype-of x)
                '(character number symbol string))))

     (defun gentype-of (o)
       "Returns the general type of the object O."
       (typecase o
         (character 'character)
         (number 'number)
         (symbol 'symbol)
         (string 'string)))
```

page 316, lines -15 – -11: Move 1 line down to change

`(let* (` {*symbol* | (*symbol form*)}`*)` special form
`[(declare (special` *variables*`))]`
forms`)` Exactly like `let` except that the assigning of initial values to the *variable*s is done in order so that one variable may be used in the initialization *form* of a later *variable.*

to

`(let* (` {*symbol* | (*symbol form*)}`*)` special form
`[(declare (special` *variables*`))]`
forms`)`
Exactly like `let` except that the assigning of initial values to the *variable*s is done in order so that one variable may be used in the initialization *form* of a later *variable.*

page 317, lines -6 – -4: Change font of four symbols from `tt` font to *italics* font, changing
`((objects1) forms1)`
...
`((objectsn) formsn))`
to
`((`*objects1*`)` *forms1*`)`
...
`((`*objectsn*`)` *formsn*`))`

page 318, line 2: Change font of "`objectsn`" to "*objectsn*".

page 318, line 17: Change font of four symbols from `tt` to *italics,* changing
`(typecase` *form* `(type1 forms1) ... (typen formsn))`
to
`(typecase` *form* `(`*type1 forms1*`) ... (`*typen formsn*`))`

page 319, lines 1–2: Insert a new one-line paragraph:
"In these three macros, a *statement* is a non-atomic (that is, a list) form."

page 319, lines 2–16: Change seven (7) occurrences of the plural italic "*forms*" to "*statements*".

page 329, line 1: Change "tt #'eql" to "`#'eql`".

page 331, line -9: Change "*#'eql*" to "`#'eql`".

page 335, line -8: Change the font of "objects" to "*objects*".

page 335, line -5: Change the font of "objects" to "*objects*".

page 336, lines 2 – -8: Change the font of eight (8) occurrences of "object" and of one occurrence of "`object`" to "*object*".

page 345: Add, as new entry:

`(ed [`*symbol*`])` function
Invokes the editor. If *symbol* is included, tries to let you edit the function named *symbol*, which might involve loading the file where it is defined into the editor. This function might not be implemented.

page 350, column 1, line 5: Insert a page number, changing "`defvar`, 176, 342" to "`defvar`, 176, 188, 342".

page 350, column 1, line -7: Insert a page number, changing "`ed`, 244" to "`ed`, 244, 345"

page 351, column 1, line -6: Insert a page number, changing "directives, 221, 222" to "directives, 221, 222, 226".

page 353, column 1, line -6: Change "McCarthy, John, 21" to "McCarthy, John, 22".